

# UD1 . Fundamentos de programación II

Verónica Mascarós

**Curso 23-24**



```
e = m(b, " ");  
-1 < e && b.splice(e, 1);  
e = m(b, void 0);  
-1 < e && b.splice(e, 1);  
e = m(b, "");  
-1 < e && b.splice(e, 1);  
for (c = 0; c < d && c < b.length;  
    a += b[c].b + ", ", n.push(  
}  
for (g = 0; g < f;) {  
    e = Math.floor(b.length *  
    d.c + "</span></li>"), b[e]  
}  
for (; c < b.length; c++) {  
    void 0 !== b[c] && ("para  
}  
function(b);  
    "single").h("mode_9  
    ent(
```

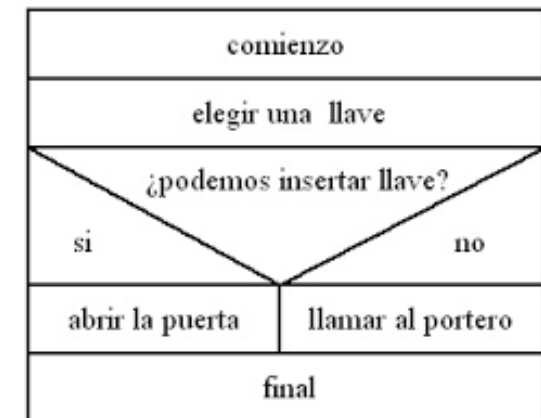
# Algoritmos

- Como ya vimos, un ALGORITMO es un conjunto ordenado y finito de operaciones que permiten resolver un problema.
- Ejemplo: Algoritmo para preparar café
  - Inicio
  - 1. BUSCAR LA CAFETERA, EL AGUA Y EL CAFÉ.
  - 2. ABRIR LA CAFETERA Y LLENAR EL DEPÓSITO DE AGUA.
  - 3. VIGILAR EL AGUA.
  - 4. PONER CAFÉ.
  - 5. CERRAR CAFETERA.
  - 6. ENCEDER EL FUEGO
  - 7. PONER LA CAFETERA A CALENTAR.
  - 8. ESPERAR HASTA QUE SALGA TODO EL CAFÉ
  - 9. RETIRAR LA CAFETERA DEL FUEGO.
  - Fin



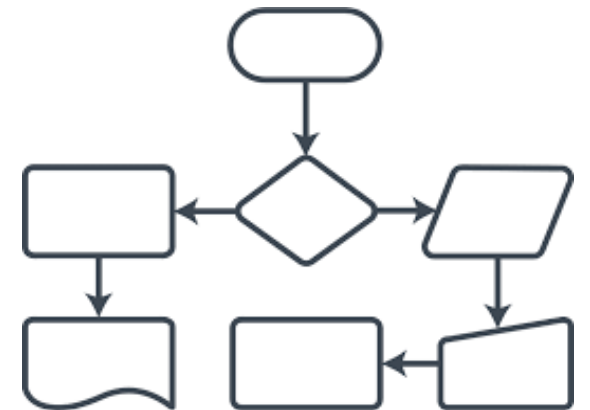
# Antes de empezar a programar...

- Las técnicas de representación de algoritmos son muy útiles para practicar y aprender a pensar de forma algorítmica antes de programar utilizando lenguajes de programación reales como Java, Python, etc.
- Veremos:
  - Diagramas de Flujo
  - Pseudocódigo
- Otras formas de representación:
  - Diagrama Nassi-Shneiderman



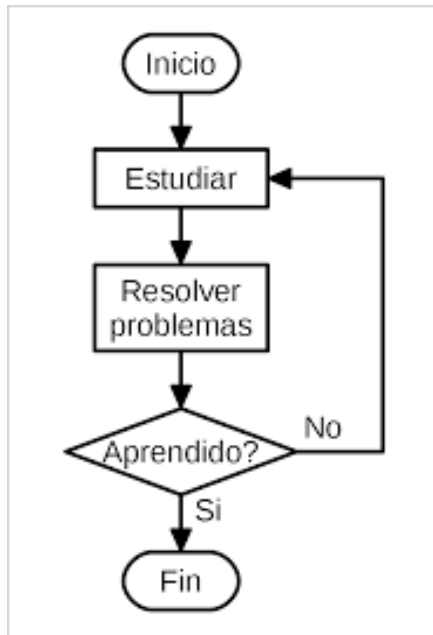
# Diagramas de flujo

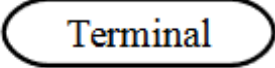



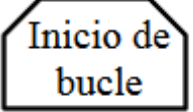
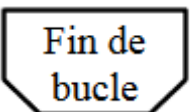
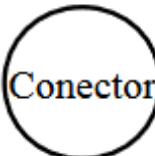
- Una **técnica gráfica** de representar algoritmos es la utilización de diagramas de flujo, flujogramas o «flowchart».
- Dicho de otro modo, representa las instrucciones paso a paso que se ejecutarían en un programa de ordenador
- Estos diagramas se componen de una serie de símbolos, con indicaciones en su interior, interconectados mediante flechas que indican el camino del programa.



# Diagramas de flujo

- Algunos de los símbolos son:



	Indica el inicio o final del algoritmo
	Simboliza las acciones a realizar por parte del equipo programado.
	Representa la entrada de datos desde el exterior al equipo programado o la salida
	Se utiliza cuando es necesario tomar decisiones que impliquen la realización de diferentes instrucciones
	Representan el principio y el final de un bucle. Dentro de un bucle se encuentran las instrucciones que deben repetirse mientras se dé alguna condición
	
	Los conectores se utilizan para unir partes del programa que no quepan en la página en la que se está realizando la representación del algoritmo. Dentro del conector se suele indicar una letra mayúscula que debe coincidir en las dos partes que están unidas.

# Diagramas de flujo

- INSTRUCCIONES DE INICIO Y FIN
- Todos los programas tienen un punto inicial y un punto final que marcan cuándo empieza y acaba un programa. En un ordinograma se representa mediante INICIO y FIN.



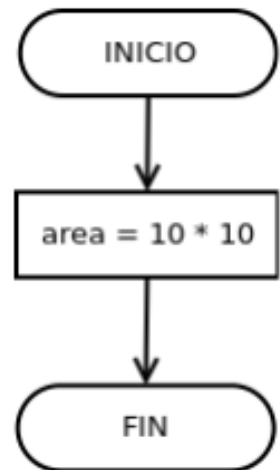
# Diagramas de flujo

- INSTRUCCIONES DE PROCESADO DE INFORMACIÓN
- Todos los programas informáticos procesan información, es decir, realizan operaciones matemáticas para calcular, manipular o modificar información.
  - Como ya vimos, las operaciones pueden ser aritméticas (+ - \* / %), relacionales (< <= > >= == <>) o lógicas (NOT, AND, OR). A su vez, se utilizan variables (A, B, Edad, Precio, etc.) para almacenar la información que estamos procesando.
- El encargado de realizar este tipo de operaciones es el procesador de un ordenador.
- En un ordinograma se representan utilizando **rectángulos** que contienen las instrucciones.

# Diagramas de flujo

- INSTRUCCIONES DE PROCESADO DE INFORMACIÓN
- Por ejemplo, el siguiente ordinograma calcula el área de un cuadrado de lado 10.

## ORDINOGRAMA



## EXPLICACIÓN

Calcula  $10 * 10$  y guarda el resultado (100) en la variable **area**.



# Diagramas de flujo

- **VARIABLES**
- Es bastante habitual que necesitemos que el programa guarde cierta información durante su ejecución. Para ello se utilizan las **variables**.
- Existen diferentes tipos de variables en función del tipo de información que se necesite conservar (variables de texto, de números enteros, de números en coma flotante, etc.).
- La **asignación** del valor a la variable se suele realizar en pseudocódigo utilizando el **símbolo igual** y es posible operar con ellas.

# Diagramas de flujo

- INSTRUCCIONES DE PROCESADO DE INFORMACIÓN
- Podemos utilizar tantas instrucciones de procesamiento de información como necesitemos. Por ejemplo, el siguiente ordinograma calcula el número de horas que hay en 10 años.

## ORDINOGRAMA



## EXPLICACIÓN

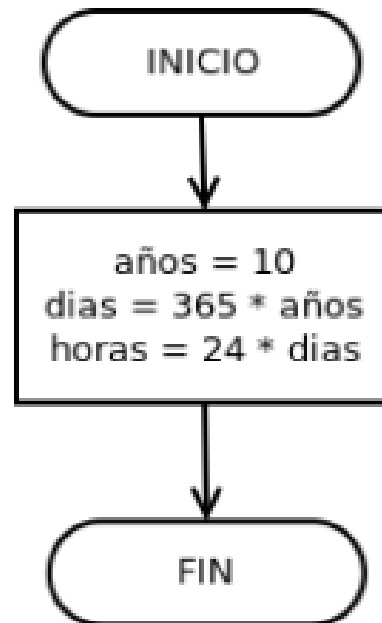
Guardamos 10 en la variable **años**.

Calcula  $365 * \mathbf{años}$  ( $365 * 10$ ) y guarda el resultado (3650) en la variable **días**.

Calcula  $24 * \mathbf{días}$  ( $24 * 3650$ ) y guarda el resultado (87600) en la variable **horas**.

# Diagramas de flujo

- INSTRUCCIONES DE PROCESADO DE INFORMACIÓN
- Si se desea, es posible poner varias instrucciones dentro de un mismo rectángulo (para simplificar)



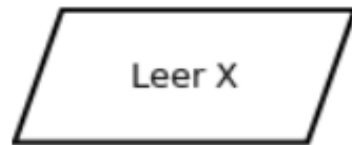
# Diagramas de flujo

- INSTRUCCIONES DE ENTRADA/SALIDA DE INFORMACIÓN
- Todos los programas informáticos utilizan algún tipo de entrada y salida de información.
- No tendría sentido crear programas sin entrada ni salida ya que solo realizarían cálculos sin comunicarse con el exterior.
- Es una parte muy importante de la programación ya que, entre otras cosas, es lo que permite a un usuario interactuar con un programa. Por ejemplo:
  - 1. El usuario introduce información por teclado (entrada).
  - 2. El programa procesa la información (hace algún cálculo).
  - 3. El programa muestra el resultado por pantalla (salida).

# Diagramas de flujo

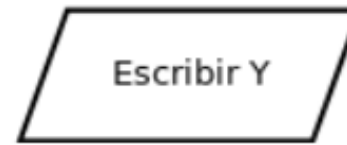
- INSTRUCCIONES DE ENTRADA/SALIDA DE INFORMACIÓN
- Por ahora nos vamos a centrar en la entrada y salida más sencilla: el teclado y la pantalla.
- En los ordinogramas la entrada y salida de información se representa mediante un paralelogramo.

## **ENTRADA de información**



El usuario introduce información por teclado y se guarda en la variable X.

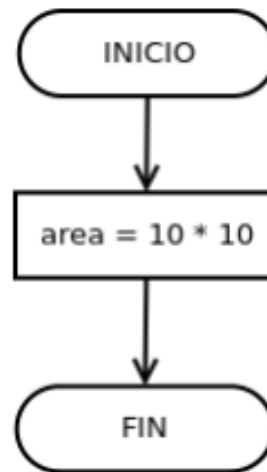
## **SALIDA de información**



Muestra por pantalla el contenido de la variable Y.

# Diagramas de flujo

- Modificad el siguiente organigrama añadiendo entrada y salida de modo que primero le pida al usuario que introduzca el lado del cuadrado, luego calcule el área, y por último la muestre.

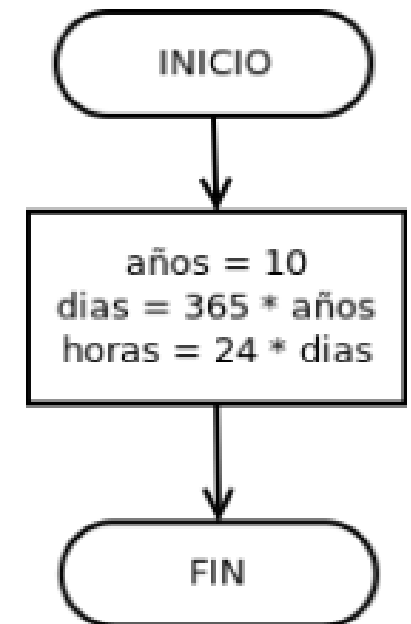


Si creáramos un programa siguiendo el algoritmo mejorado (con entrada y salida) **nos permitiría utilizarlo para calcular el área de cualquier cuadrado.** Esto es mucho más útil y general que el programa original que solo calcula el área de un cuadrado de lado 10. Por ello, **añadir entrada y salida a los programas es fundamental** para que sean de utilidad.

# Diagramas de flujo

- Modificad el organigrama que calculaba los días y horas para añadirle entrada y salida. Queremos que el programa le pida al usuario que introduzca por teclado los años, calcule los días y las horas, y lo muestre por pantalla.

Si creáramos un programa que siguiera los pasos de este ordinograma, lo podríamos utilizar para calcular el número de horas de los años que quisiéramos, tantas veces como quisiéramos.



# Diagramas de flujo

- ESTRUCTURAS DE CONTROL
- Hasta ahora hemos visto algoritmos en los que las instrucciones se ejecutan secuencialmente (una después de la otra).
- Pero a menudo es necesario diseñar algoritmos cuyas instrucciones no se ejecuten secuencialmente, para lo que es necesario utilizar estructuras de control.
- Las **estructuras de control** son utilizadas para **controlar la secuencia** (el orden) en el que se ejecutan las instrucciones.



# Diagramas de flujo

- ESTRUCTURAS DE CONTROL
- Existen dos tipos:
  - Estructuras alternativas: Permiten alternar entre distintas instrucciones (ejecutar unas u otras) dependiendo de una condición. Pueden ser simples, dobles o múltiples.
  - Estructuras repetitivas: Permiten repetir instrucciones (ejecutarlas varias veces)

# Diagramas de flujo

- ESTRUCTURAS ALTERNATIVAS
- Controlan la ejecución/no ejecución de una o más instrucciones en función de que se cumpla una condición.
  - Dicho de otra manera, se utilizan para que sucedan cosas distintas dependiendo de una condición.
- Por ejemplo, al introducir una contraseña si esta es correcta se iniciará sesión, pero si es incorrecta mostrará un mensaje de error.

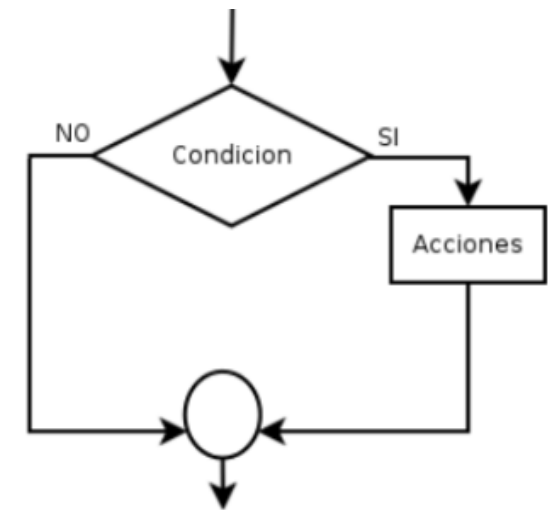
# Diagramas de flujo

- ESTRUCTURAS ALTERNATIVAS
- Existen tres tipos de estructuras alternativas: Simple, Doble y Múltiples.
- Todas ellas utilizan condiciones, como (precio > 200), (edad >= 18), (contraseña == "1234"), etc.
- En un ordinograma una condición se expresa mediante un rombo.



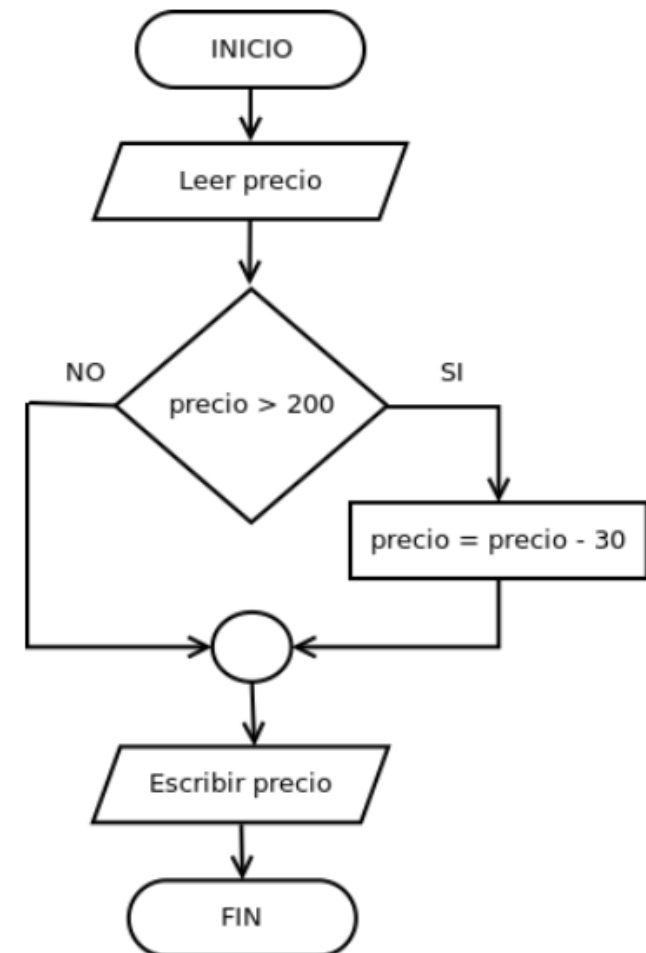
# Diagramas de flujo

- ESTRUCTURAS ALTERNATIVAS SIMPLES
- Si la condición es verdadera se ejecutará una o varias instrucciones concretas, pero si es falsa éstas no se ejecutarán.
- Se representa así:



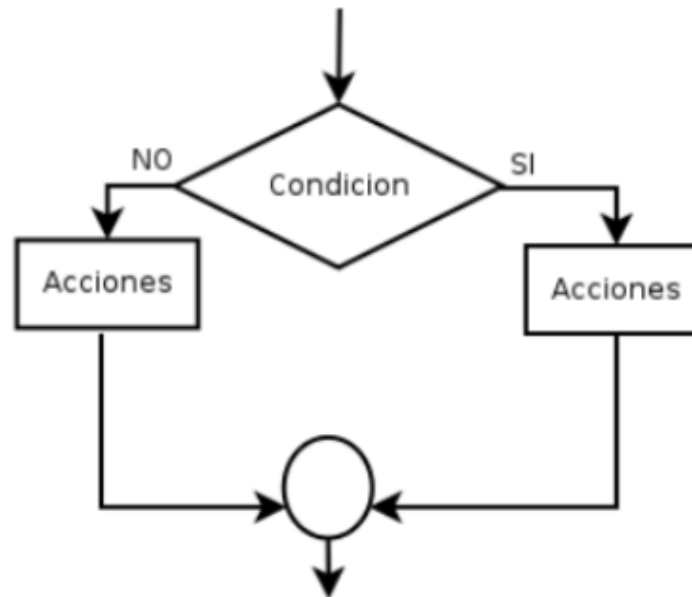
# Diagramas de flujo

- ESTRUCTURAS ALTERNATIVAS SIMPLES
- Un programa que lee por teclado el precio inicial de un producto, si es superior a 200 € se le aplica un descuento de 30 €, y por último lo muestra por pantalla



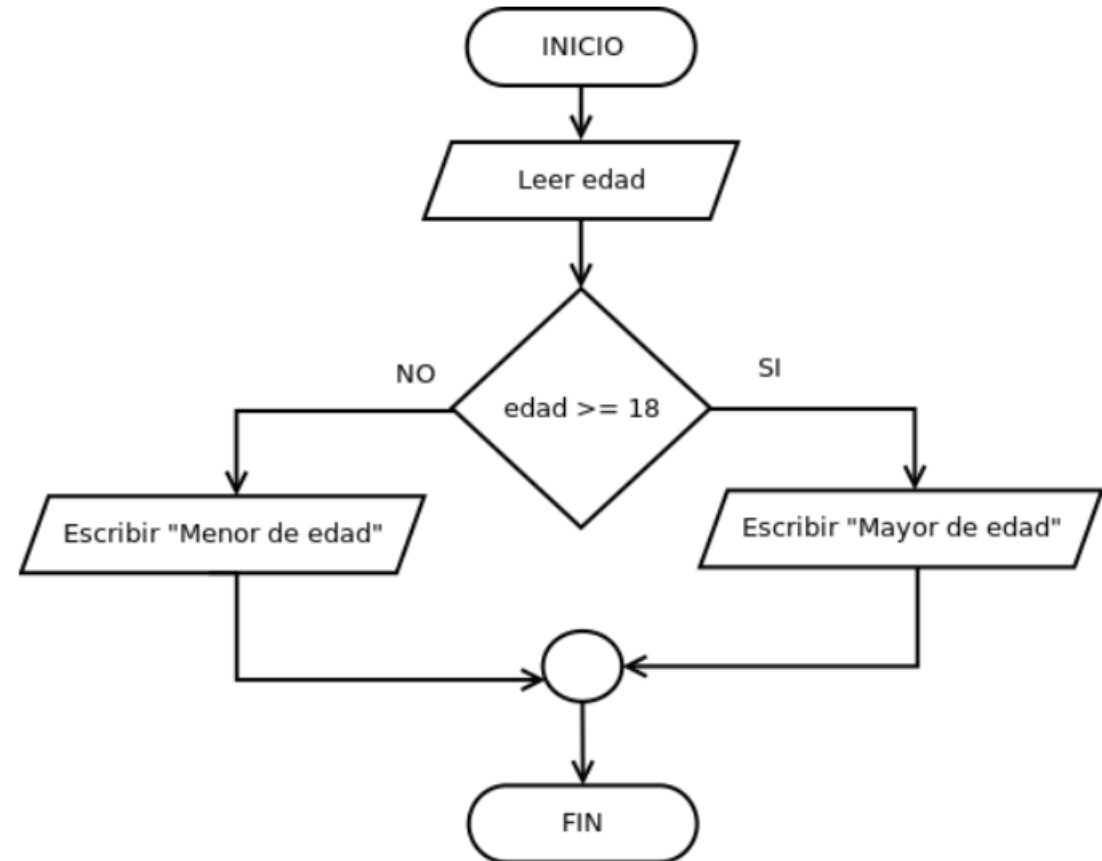
# Diagramas de flujo

- ESTRUCTURAS ALTERNATIVAS DOBLES
- Si la condición es cierta se ejecutarán unas instrucciones y si es falsa se ejecutarán otras distintas.



# Diagramas de flujo

- ESTRUCTURAS ALTERNATIVAS DOBLES
- Ejemplo: Un programa que pide la edad por teclado, si es mayor o igual a 18 mostrará por pantalla el texto "Mayor de edad", en caso contrario mostrará "Menor de edad".



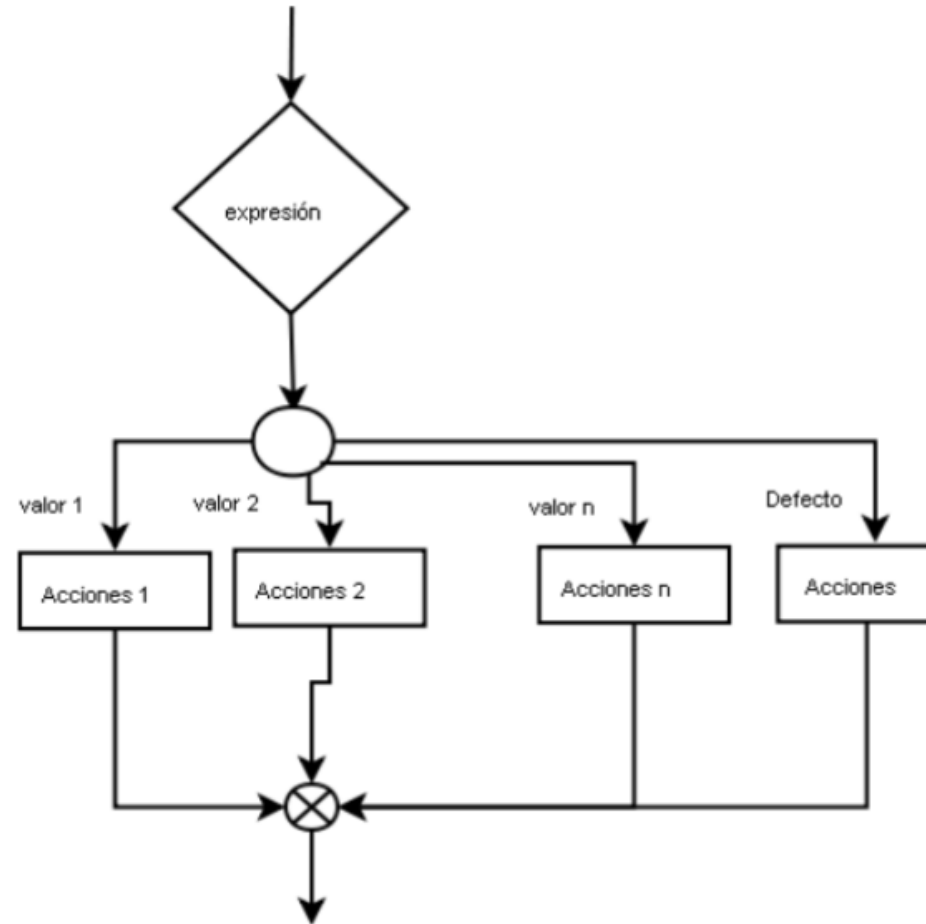
# Diagramas de flujo

- ESTRUCTURAS ALTERNATIVAS MÚLTIPLE
- Se pueden especificar distintas instrucciones dependiendo del
- valor de una expresión. ¡Ojo! de una expresión, no de una condición.
- La expresión dará un resultado concreto (por ejemplo 10, -5, 0.25, etc.) y se ejecutarán las instrucciones asociadas a cada valor.
- Si el resultado de la expresión no es igual a ninguno de los valores indicados, se ejecutarán las instrucciones por defecto (si se ha indicado, esto es opcional).



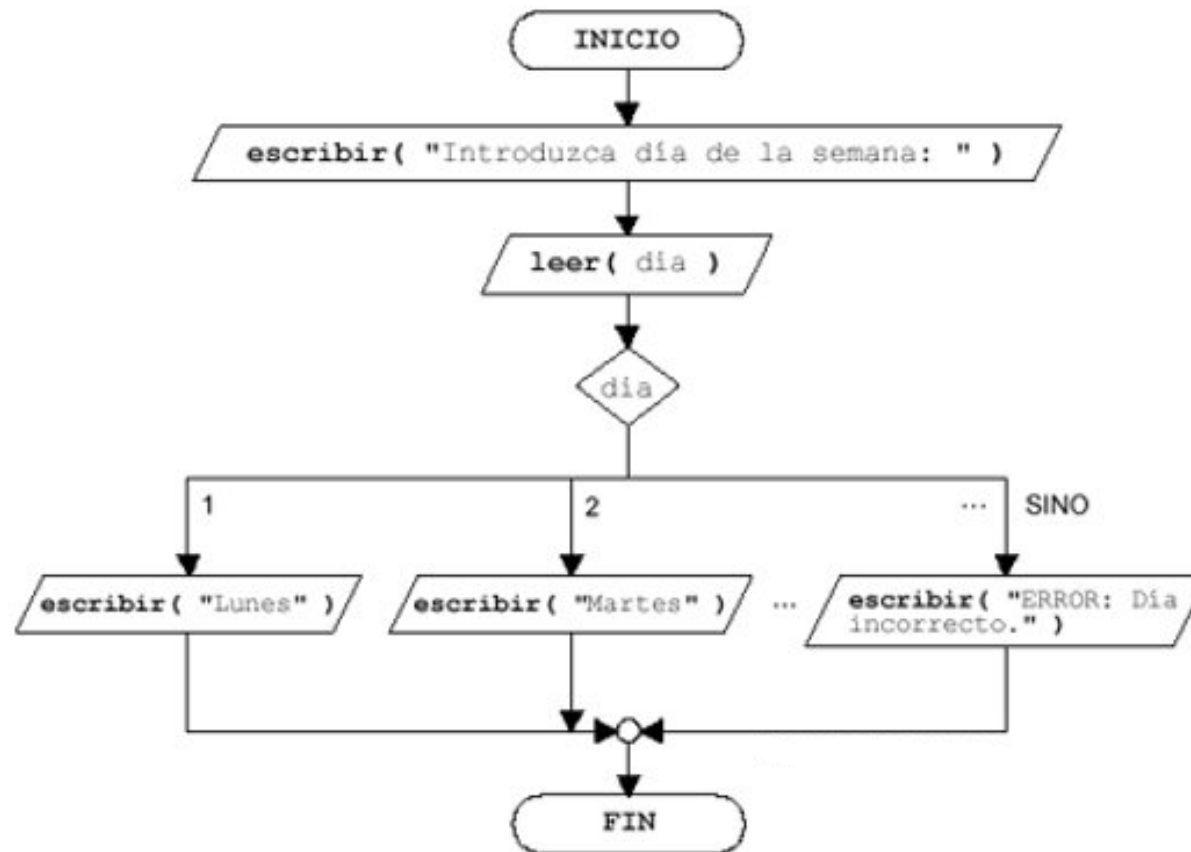
# Diagramas de flujo

- ESTRUCTURAS ALTERNATIVAS MÚLTIPLE
- Estructura



# Diagramas de flujo

- ESTRUCTURAS ALTERNATIVAS MÚLTIPLE
- Ejemplo



# Diagramas de flujo

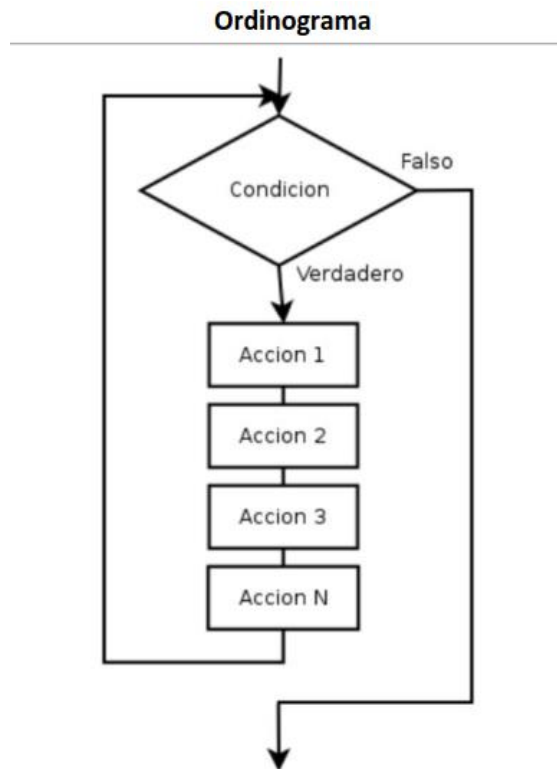
- ESTRUCTURAS REPETITIVAS
  - Las instrucciones repetitivas (o bucles) son aquellas que permiten variar o alterar la secuencia normal de ejecución de un programa haciendo posible que un grupo de operaciones (acciones) se repita un número determinado o indeterminado de veces, dependiendo del cumplimiento de una condición.
  - Veremos tres tipos: Bucle Mientras (WHILE), Bucle Hacer-Hasta (DO-WHILE) y Bucle Para (FOR)

# Diagramas de flujo

- ESTRUCTURAS REPETITIVAS - WHILE
- En la estructura Mientras o "WHILE" el bloque de acciones se repite **mientras la condición sea cierta**, evaluándose siempre la condición antes de entrar en el bucle, por ello **es posible que las acciones no se ejecuten nunca**.

# Diagramas de flujo

- ESTRUCTURAS REPETITIVAS - WHILE



## Pseudocódigo

Mientras Condición hacer

    Instrucción 1

    Instrucción 2

    ...

    Instrucción N

FinMientras

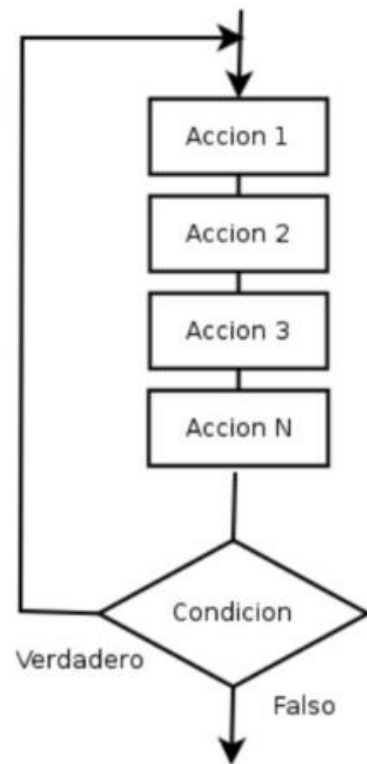
# Diagramas de flujo

- ESTRUCTURAS REPETITIVAS - DO WHILE
- En la estructura hasta o "DO-WHILE", el bloque de instrucciones se repite **mientras que la condición sea cierta**, y la condición se evalúa al final del bloque por lo que siempre se ejecutarán **al menos una vez** el bloque de instrucciones

# Diagramas de flujo

- ESTRUCTURAS REPETITIVAS - DO WHILE

Ordinograma



Pseudocódigo

```
Repetir  
    Instrucción 1  
    Instrucción 2  
    ...  
    Instrucción N  
Mientras Condición.
```

# Diagramas de flujo

- ESTRUCTURAS REPETITIVAS - FOR
- En la estructura Para o "FOR" se conoce de antemano el número de veces que se ejecutará el
- bloque de instrucciones.
- El bloque de acciones se repite mientras que la condición sea cierta, evaluándose siempre la condición antes de entrar en el bucle, por ello es posible que las acciones no se ejecuten nunca.
- Esta explicación es idéntica a la del bucle WHILE, pero un bucle FOR debe cumplir las siguientes características:
  1. La variable contador se inicializa con un valor inicial.
  2. La condición siempre debe ser:  $\text{variable\_contador} \leq \text{valor\_final}$
  3. En cada interacción, la variable contador se incrementa en un determinado valor.



# Diagramas de flujo

- ESTRUCTURAS REPETITIVAS - FOR

## Pseudocódigo

Para Var\_Cont de ValorInicial a ValorFinal con  
Incremento = n

    Instrucción 1

    Instrucción 2

    ...

    Instrucción N

FinPara

## Ordinograma

